APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: **METHOD AND APPARATUS FOR INTERPOSING KERNEL SYMBOLS**

APPLICANT: **Chris J. KIICK, Michael T. CLAYTON, and Michael S. WALKER**

32615
PATENT TRADEMARK OFFICE

"EXPRESS MAIL" Mailing Label Number: **EV436028298US**

Date of Deposit: **March 12, 2004**

METHOD AND APPARATUS FOR INTERPOSING KERNEL SYMBOLS

Background

[0001] A kernel is the essential center of a computer operating system, the core that provides basic services for all other parts of the operating system. Typically, the kernel includes an interrupt handler that handles all requests or completed input/output (I/O) operations that compete for the kernel's services, a scheduler that determines which programs share the kernel's processing time, etc. A kernel's services are requested by other parts of the operating system or by application through a specified set of program interfaces sometimes known as system calls. The kernel is typically associated with a symbol table. The symbol table provides an association of a kernel symbol and a kernel symbol definitions. In addition, the kernel typically includes functionality to resolve kernel symbols, *i.e.,* determine which kernel symbol definition to keep if a given kernel symbol is defined multiple times.

[0002] In some instances, the functionality of the kernel may be augmented by the addition of kernel modules. The kernel modules are typically added after the kernel has booted using a kernel runtime loader. When the kernel module is loaded, the kernel runtime loader examines the static dependencies of the kernel module and loads the appropriate kernel modules upon which the kernel module being loaded depends. Each time a kernel module is loaded and prior to loading a subsequent kernel module, the symbols present in the kernel module are resolved against the kernel symbol table.

[0003] Once all the kernel modules upon which the particular kernel module depends have been loaded, the kernel module is loaded. If the dependencies of a

given kernel module change, the kernel module is typically recompiled with the new dependencies and subsequently re-loaded into the kernel.

Summary

[0004] In general, in one aspect, the invention relates to a method for loading a kernel module, comprising loading a preload module into a kernel, wherein loading the preload module comprises creating a dynamic dependency list, loading the kernel module comprising a static dependency list into the kernel, triggering a hook when the static dependency list is reviewed, obtaining module information associated with the kernel module using the hook, searching for a dynamic dependency associated with the kernel module in the dynamic dependency list using the module information, and updating the static dependency list with the dynamic dependency to obtain an updated static dependency list if the dynamic dependency associated with the kernel module is present in the dynamic dependency list.

[0005] In general, in one aspect, the invention relates to a system comprising a kernel module having a static dependency list and a kernel symbol, a modified kernel runtime loader configured to load the kernel module using the static dependency list, and a preload module comprising a dynamic dependency list, wherein the preload module is configured to install a hook into the modified kernel runtime loader, wherein the hook is configured to update the static dependency list using the dynamic dependency list to obtain an updated static dependency list if a dynamic dependency associated with the kernel module is present in the dynamic dependency list.

[0006] In general, in one aspect, the invention relates to a computer system for loading a kernel module, comprising a processor, a memory, a storage device, a computer display, and software instructions stored in the memory for enabling the

computer system under control of the processor, to perform loading a preload module into a kernel, wherein loading the preload module comprises creating a dynamic dependency list, loading the kernel module comprising a static dependency list into the kernel, triggering a hook when the static dependency list is reviewed, obtaining module information associated with the kernel module using the hook, searching for a dynamic dependency associated with the kernel module in the dynamic dependency list using the module information, and updating the static dependency list with the dynamic dependency to obtain an updated static dependency list, if the dynamic dependency associated with the kernel module is present in the dynamic dependency list.

[0007] In general, in one aspect, the invention relates to a network system having a plurality of nodes, comprising a kernel module having a static dependency list and a kernel symbol, a modified kernel runtime loader configured to load the kernel module using the static dependency list, and a preload module comprising a dynamic dependency list, wherein the preload module is configured to install a hook into the modified kernel runtime loader, wherein the hook is configured to update the static dependency list using the dynamic dependency list to obtain an updated static dependency list if a dynamic dependency associated with the kernel module is present in the dynamic dependency list, wherein the kernel module executes on any node of the plurality of nodes, wherein the modified kernel module runtime loader executes on any node of the plurality of nodes, wherein the preload module executes on any node of the plurality of nodes.

[0008] Other aspects of the invention will be apparent from the following description and the appended claims.

Brief Description of Drawings

[0009] Figure 1 shows a flow diagram in accordance with one embodiment of the invention.

[0010] Figure 2 shows a flow chart in accordance with one embodiment of the invention.

[0011] Figure 3 shows a networked computer system in accordance with one embodiment of the invention.

Detailed Description

[0012] Specific embodiments of the invention will now be described in detail with reference to the accompanying figures. Like elements in the various figures are denoted by like reference numerals for consistency.

[0013] In the following detailed description of embodiments of the invention, numerous specific details are set forth in order to provide a more thorough understanding of the invention. However, it will be apparent to one of ordinary skill in the art that the invention may be practiced without these specific details. In other instances, well-known features have not been described in detail to avoid obscuring the invention.

[0014] In general, embodiments of the invention relate to a method and apparatus for dynamically changing kernel module dependencies. More specifically, embodiments of the invention relate to a method and apparatus for dynamically changing module dependencies without recompiling the kernel module. Further, embodiments of the invention provide a method for interposing of kernel symbols by allowing dependencies of a kernel module to be dynamically updated to depend from an interposing kernel module. Embodiments of the invention also allow multiple definitions of a kernel symbol to operate within the kernel.

[0015] Figure 1 shows a flow diagram in accordance with one embodiment of the invention. In one or more embodiments of the invention, a kernel (100) includes a modified kernel runtime loader (102). The modified kernel runtime loader (102) typically includes functionality to review a static dependency list (104) present in a kernel module (106), which is being loaded into the kernel (100). In addition, the modified kernel runtime loader (102) also includes functionality to allow multiple definitions for a given kernel symbol. Specifically, in one embodiment of the invention, the kernel module (106) includes a flag that signals the modified kernel runtime loader (102) not to check for multiply defined kernel symbols.

[0016] The modified kernel runtime loader (102) includes a hook (108) that is triggered when the modified kernel runtime loader (102) is reviewing the static dependency list (104). In one embodiment of the invention, the hook (108) obtains the kernel module (106) information, typically a name and version for the kernel module (106), and forwards this information to a preload module (110). The preload module (110) (typically loaded after the kernel (100) has booted) includes functionality to install the hook (108) into a kernel runtime loader (not shown) to produce the modified kernel runtime loader (102). Further, the preload module (110) typically includes functionality to receive and compare the kernel module (106) information against a dynamic dependency list (112).

[0017] In one or more embodiments of the invention, the dynamic dependency list (112) includes the dynamic dependencies of various kernel modules. If the kernel module (106), as identified by the kernel module (106) information, is present on the dynamic dependencies list (112), then the preload module (110) forwards this information to the hook (108). The hook (108), which includes functionality to modify the static dependency list (104), attaches dynamic dependencies (114) to the static dependency list (104). Typically, the dynamic dependencies (114) are inserted at the beginning of the static dependency list (104) such that kernel

module (106) will load kernel modules referenced by the dynamic dependencies (114) first.

[0018] In one or more embodiment of the invention, the dynamic dependencies list (112) is obtained from a configuration file (116). The dynamic dependencies list (112) may be generated immediately after the preload module (110) is loaded into the kernel (100). In one embodiment of the invention, if the configuration file (116) changes after the preload module (110) has been loaded, the preload module (110) must be unloaded and reloaded into the kernel (100). This allows the preload module (110) to read the updated configuration file (116) and update the dynamic dependency list (112) accordingly.

[0019] In one embodiment of the invention, the preload module (110) may be used in combination with an interposer module (118) to intercept kernel function calls. The interposer module (118) may include kernel symbols to be interposed. In one embodiment of the invention, the preload module (110) may specify that the kernel module (106) is dependent on the interposer module (118), thereby forcing the kernel (100) to load the interposer module (118) and allowing the interposer module (118) to redefine one or more kernel symbols used by the kernel module (106) prior to loading the kernel module (106). Thus, when the kernel symbols in the kernel module are resolved, the kernel module uses the kernel symbol definitions specified by the interposer module (118) as opposed to the symbol definitions specified by the kernel (100). Thus, the interposer module (118) allows the kernel symbols to be redefined and subsequently used during symbol resolution of subsequently loaded kernel modules (106).

[0020] In one embodiment of the invention, the interposer module (118) includes a reference count that prevents the interposer module (118) from being unloaded while a kernel module (106) that is dependent on the interposer module (118) is still loaded. In one embodiment of the invention, the reference count may be an

integer where the value represents the number of kernel modules (106) that are dependent on the interposer module (118). Each time a kernel module (106) that is dependent on the interposer module (118) is loaded into the kernel (100), the reference count is incremented by one. Similarly, each time a kernel module (106) that is dependent on the interposer (118) module is unloaded from the kernel (100), the reference count is decreased by one. The interposer module (118) may only be unloaded from the kernel (100) when the reference count equals zero.

[0021] In one embodiment of the invention, the syntax of the configuration file may include one or more lines delimited by semicolons, where each line designates one kernel module to load along with the kernel modules that depend on the kernel module. Each line in the configuration file may include the kernel module name followed by a colon and then one or more dependent kernel module names terminated by a semi-colon. Kernel modules (dependent or otherwise) may be listed more than once as dependencies are additive.

[0022] In addition, the syntax of the configuration file may be used to reverse the meaning of the dependency list discussed above. Specifically, the syntax may indicate to the modified kernel runtime loader that a given kernel module is always to be loaded prior to loading any kernel module unless a particular kernel module is being loaded. This functionality may be denoted in the syntax using a "*" (quotes may be included in the configuration file for correct parsing, as required).

[0023] The following examples illustrate how the aforementioned syntax may be used, note that the text following the # corresponds to a comment.

Configuration File Examples

```
# Example 1
foo:bar;    # kernel module bar is dependent on kernel module foo

# Example 2
prefs   :
```

```
           myfs
           yourfs
           hisfs;
# if any of myfs, yourfs or hisfs is loaded, prefs must be loaded first.

#Example 3
    foo : complex;
    bar : complex;
    baz : complex;
# when loading kernel module complex, kernel modules foo bar and baz must be
loaded first.

#Example 4
    thin :  tall fat;
    short : tall fat ;
# modules tall and fat depend on modules thin and short

#Example 5
raboof:      "*"
      foo bar baz;
#for every kernel module EXCEPT foo, bar and baz, load kernel module raboof
```

[0024] In one embodiment of the invention, if a syntax error is encountered, the preload module stops reading the configuration file. However, the dynamic dependencies successfully read from the configuration file prior to the error is placed in the dynamic dependencies list within the preload module.

[0025] Figure 2 shows a flow chart in accordance with one embodiment of the invention. Initially, a configuration file defining dynamic dependencies is generated (Step 100). Embodiments of the configuration file may use the syntax described above. The kernel is subsequently booted and a symbol table initialized and populated (Step 102). The configuration file may be defined after the kernel is booted, however, the preload module must be loaded after the configuration file is defined.

[0026] The preload module is then loaded into the kernel and, under the direction of the preload module, a hook is installed into the kernel runtime loader to generate a modified kernel runtime loader (Step 104). In addition, when the preload module is loaded, the preload module reads the configuration file into an

internal representation of the configuration file (*e.g.*, a dynamic dependencies list). After the kernel has been booted and the preload module has been loaded, a request may be received by the kernel to load an additional kernel module into the kernel (Step 106).

[0027] The request to load the additional kernel module into the kernel triggers the modified kernel runtime loader to review the static dependency list in the additional kernel module (Step 108). The review of the static dependency list triggers a hook in the modified kernel runtime loader (Step 110). The hook, in one embodiment of the invention, includes a pointer to the additional kernel module and a call back argument. The hook subsequently obtains the additional kernel module information, typically the name and version of the additional kernel module, and calls into the preload module to determine whether there are any dynamic dependencies associated with the additional kernel module exist (Step 112).

[0028] If no dynamic dependencies associated with the additional kernel module, the hook returns control back to the kernel runtime loader which proceeds to load the kernel modules (as required) defined in the additional kernel modules static dependency list (Step 114). The kernel symbols of the loaded kernel modules upon which the additional kernel module depends are subsequently resolved (Step 116). After all the kernel modules upon which the additional kernel module is dependent are loaded and their kernel symbols resolved, the additional kernel module is loaded (Step 118) and the kernel symbols within the additional kernel module are resolved (Step 120).

[0029] If there are dynamic dependencies associated with the additional kernel module (Step 112), then the hook obtains the dynamic dependencies from the dynamic dependency list and adds them to the static dependency list thereby updating the static dependency list (Step 122). In one embodiment of the

invention, the dynamic dependencies are added to the front of the static dependency list thereby forcing the kernel modules listed in the dynamic dependencies list to be loaded before the kernel modules listed in the static dependencies list. The aforementioned functionality may allow an interposer module to be loaded prior to the additional kernel module. The interposer module may then redefine kernel symbols, which are then subsequently used by the additional kernel module. Continuing with Figure 2, once the static dependency list has been updated with the dynamic dependencies listed in the dynamic dependency list, Steps 114-120 (listed above) are performed.

[0030] As noted above, the kernel module may include a flag that signals the modified kernel runtime loader that multiple kernel symbol definitions are allowed. Though not shown in Figure 2, one skilled in the art will appreciate that kernel runtime loader may look for a flag to indicate that multiple kernel symbol definitions are allowed prior to steps 116 and 120.

[0031] In general, in one aspect, the invention relates to a method for loading a kernel module, comprising loading a preload module into a kernel, wherein loading the preload module comprises creating a dynamic dependency list, loading the kernel module comprising a static dependency list into the kernel, triggering a hook when the static dependency list is reviewed, obtaining module information associated with the kernel module using the hook, searching for a dynamic dependency associated with the kernel module in the dynamic dependency list using the module information, and updating the static dependency list with the dynamic dependency to obtain an updated static dependency list if the dynamic dependency associated with the kernel module is present in the dynamic dependency list.

[0032] In general, in one aspect, the invention relates to a system comprising a kernel module having a static dependency list and a kernel symbol, a modified

kernel runtime loader configured to load the kernel module using the static dependency list, and a preload module comprising a dynamic dependency list, wherein the preload module is configured to install a hook into the modified kernel runtime loader, wherein the hook is configured to update the static dependency list using the dynamic dependency list to obtain an updated static dependency list if a dynamic dependency associated with the kernel module is present in the dynamic dependency list.

[0033] In general, in one aspect, the invention relates to a computer system for loading a kernel module, comprising a processor, a memory, a storage device, a computer display, and software instructions stored in the memory for enabling the computer system under control of the processor, to perform loading a preload module into a kernel, wherein loading the preload module comprises creating a dynamic dependency list, loading the kernel module comprising a static dependency list into the kernel, triggering a hook when the static dependency list is reviewed, obtaining module information associated with the kernel module using the hook, searching for a dynamic dependency associated with the kernel module in the dynamic dependency list using the module information, and updating the static dependency list with the dynamic dependency to obtain an updated static dependency list, if the dynamic dependency associated with the kernel module is present in the dynamic dependency list.

[0034] In general, in one aspect, the invention relates to a network system having a plurality of nodes, comprising a kernel module having a static dependency list and a kernel symbol, a modified kernel runtime loader configured to load the kernel module using the static dependency list, and a preload module comprising a dynamic dependency list, wherein the preload module is configured to install a hook into the modified kernel runtime loader, wherein the hook is configured to update the static dependency list using the dynamic dependency list to obtain an updated static dependency list if a dynamic dependency associated with the kernel

module is present in the dynamic dependency list, wherein the kernel module executes on any node of the plurality of nodes, wherein the modified kernel module runtime loader executes on any node of the plurality of nodes, wherein the preload module executes on any node of the plurality of nodes.

[0035] The invention may be implemented on virtually any type of computer regardless of the platform being used. For example, as shown in Figure 3, a networked computer system (300) includes a processor (302), associated memory (304), a storage device (306), and numerous other elements and functionalities typical of today's computers (not shown). The networked computer (300) may also include input means, such as a keyboard (308) and a mouse (310), and output means, such as a monitor (312). The networked computer system (300) is connected to a local area network (LAN) or a wide area network (*e.g.*, the Internet) (not shown) via a network interface connection (not shown). Those skilled in the art will appreciate that these input and output means may take other forms. Further, those skilled in the art will appreciate that one or more elements of the aforementioned computer (300) may be located at a remote location and connected to the other elements over a network.

[0036] While the invention has been described with respect to a limited number of embodiments, those skilled in the art, having benefit of this disclosure, will appreciate that other embodiments can be devised which do not depart from the scope of the invention as disclosed herein. Accordingly, the scope of the invention should be limited only by the attached claims.